Building Localized Mac and iPhone Applications

Session 106

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

At the convention center, a movie will be placed here

Building Localized Mac and iPhone Applications

Lee Collins
Manager, OS Engineering Asia
J. J. Enser
Software Engineer

Why Should You Care?

- More customers
- iPhone ships in more than 80 countries
- App Store available in 77 countries
- iPhone localized in 30 languages
  - Mac localized in 18
- Platforms support content in even more

Mac-Localized Languages

iPhone-Localized Languages

International Text Content

- Create, search, browse, etc.
- All of the 30 localized languages
- Plus
  - iPhone
    - Flemish, Hungarian, Icelandic, Latvian, Lithuanian
  - Mac
    - Armenian, Azeri, Cherokee, Dari, Georgian, Gujarati, Gurmukhi, Hawaiian, Hindi, Inuktitut, Jawi, Kazakh, Kurdish, Maltese, Nepali, Pashto, Persian, Sami, Serbian, Tamil, Tibetan, Uyghur, Vietnamese, Welsh

Write Once; Run Everywhere

- Basis of iPhone and Mac design
- Single binary should support
  - Multiple localizations
  - Any language content
- Right out of the box

Internationalization and Localization

- Internationalization (i18n)
  - Prepare your software to be used in different languages and locales
  - Support for content
- Localization (l10n)
  - Add data for specific languages and locales
- Both must be done
- Internationalization should be done first
- Localization can be more easily outsourced

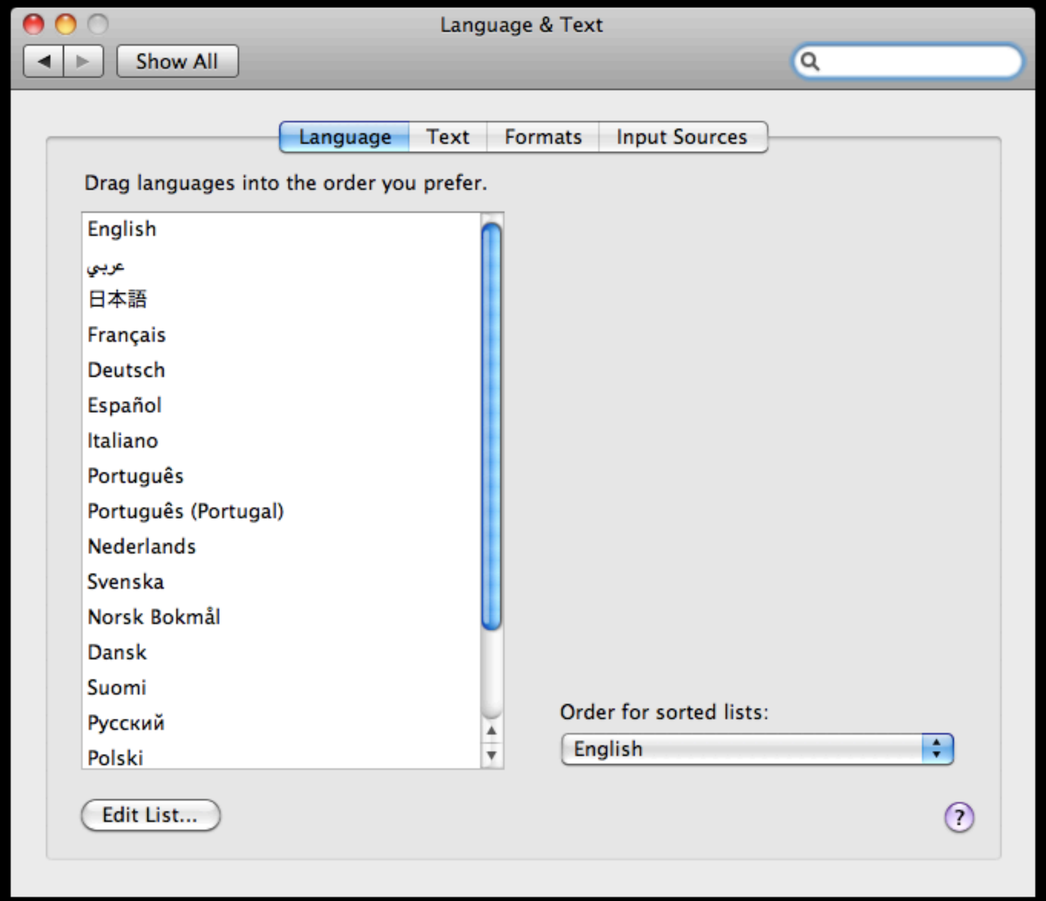
Key Ideas

- Internationalization can be difficult, but…
- Using the built-in APIs removes almost all of the difficulty
- Internationalization is easier with good design

Code You Shouldn’t Have to Write

- UI for selecting language
- Mechanism for getting localized strings, etc.
- Code for string, date-time, number format, etc.
- Different binary for each language
  - Single binary, multiple localizations

Carrier
٢:٣٨ م
General
International
Language
English
Keyboards
4
Region Format
Egypt
Region Format Example
الاثنين، ٥ يناير، ٢٠٠٩
١٢:٣٤ ص
4085551212


Language & Text
Show All
Language
Text
Formats
Input Sources
Drag languages into the order you prefer.
English
عربي
日本語
Français
Deutsch
Español
Italiano
Português
Português (Portugal)
Nederlands
Svenska
Norsk Bokmål
Dansk
Suomi
Русский
Polski
Order for sorted lists:
English
Edit List...
?

Three Steps to Internationalization

1. Use Unicode
2. Use bundles
3. Use locales

Using Unicode

Unicode

- Character set aimed at providing simultaneous support for all the world's languages
- Current version is 5.1.0 (April 2008)
- Currently covers more than 100,000 characters in 77 scripts
- Rapidly becoming the standard way to represent text on the Internet

Unicode Is Not 16-Bit ASCII!

- Far more complicated
- Key features
  - Three encoding forms
  - Byte-order mark
  - Four (five) normalization forms
  - Grapheme clusters
  - And many more!
- Don’t try to do it on your own!

Text Direction Can Vary

- Text direction varieties
  - Left-to-right
    - Hello
  - Right-to-left
    - سلام عليكم
  - Mixed
    - 120GB דיסק קשיח
- Be ready for all of the above

Character, Glyph, Grapheme

- Character ≠ glyph
  - Ligature
    - क्ष = क + ् + ष
  - Grapheme
    - ệ = e + ^ + .
  - Surrogate
    - 𠜎 = 0xD840 + 0xDCF5
- Don’t assume that Unicode is simple
  - You cannot break between arbitrary Unichars
  - Character widths are not necessarily additive

The NSString Class

- Core class for representing text
- Toll-free bridged with the CFStringRef opaque type
- Isolates you from complexities of Unicode text
- Interconvert with dozens of text encodings
- Related classes
  - NSMutableString
    - For strings that need on-the-fly manipulation
  - NSAttributedString
    - For rich text
  - NSMutableAttributedString
    - For both

Creating NSStrings

```
@”Use string constants with care”
```

- Not for localized strings

```
+(id) stringWithCharacters:(const unichar *)chars length:(NSUInteger)length
```

- Don’t use `wchar_t`!

```
+(id) stringWithCString:(const char*)cString encoding:(NSStringEncoding)enc
-(NSString*) substringWithRange:(NSRange) range
etc...
```

Formatting Strings

```
+(id) stringWithFormat:(NSString*)format,...
```

- Syntax is almost identical to that used by sprintf()

  %@ is used to include Objective-C objects in the format

- You can specify the order of the parameters

  %n$@

  - One-based indexing

Localized Strings

- Stored in `.strings` files
  - Use UTF-16 for the encoding
- Stored as key/value pairs
  - “Key” = “Value”;
- Accessed through
  - `NSLocalizedString( @”Key”, @”Localization hint” );`
    - Strings come from the “`Localizable.strings`” file

Localized String Formatting (Wrong Way)

- The wrong way

```
sentence = [NSString stringWithFormat:@"%@ %@ %@ %@ %@", name,
NSLocalizedString(@"wasFirstClimbedOn", @""), [self dateAsString:climbedDate],
NSLocalizedString(@"andHasAHeightOf", @""), [self heightAsString:height]];
```

- This is not localizable!
  - Sentence order is hardwired in code
  - Format needs to vary by language

Localized String Formatting (Correct Way)

- Separate format from items being displayed
- Formats in .strings files

```
"localizedFormat" = "%1$@ was first climbed on %3$@ and has a height of %2$@"

"localizedFormat" = "Bien que %2$@ de haut, %1$@ aient été montés la première
fois %3$@."
```

- Get the localized format

```
format = NSLocalizedString(@"localizedFormat", nil);
```

- Then format the string

```
formattedString = [NSString stringWithFormat:format, name, [self
heightAsString:height], [self dateAsString:climbedDate]];
```

Common NSString Operations

Casing	uppercaseString, lowercaseString
Comparison	localizedStandardCompare, isEqualToString
Substrings	substringWithRange, substringFromIndex, substringToIndex
Graphemes	rangeOfComposedCharacterSequenceAtIndex
Conversion	getCharacters, getCString
Find and Replace	rangeOfString, replaceOcurrencesOfString
Editing	insertString, appendString , deleteCharactersInRange

Other Classes

Tokenization	CFStringTokenizer
Number Formatting	NSNumberFormatter
Date, Time Formatting	NSDateFormatter
Character Properties	NSCharacterSet
Identifying Linguistic Content	NSOrthography
Transliteration Between Scripts	CFStringTransform
Normalization Between Unicode Forms	CFStringNormalize

genstrings

- Tool to scan your code and produce an appropriate `.strings` file template
- Inserts comments found in the code as clues to the localizer
- Run the tool over your `*.m` files
  - The tool defaults are usually the best settings

```
36
37
38
39
40
```

```
NSString *mountain = NSLocalizedString(@"tallMountain",
                                       @"Name of the tallest mountain");
```

bash bash

```
dsyu-mbp: tmp  dsyu$  ls
Moutains.m
dsyu-mbp: tmp dsyu$  genstrings  Mountains.m
dsyu-mbp: tmp dsyu$ ls
Localizable.strings       Mountains.m
dsyu-mbp: tmp  dsyu$
```

Using Bundles

Using Bundles

- Bundles are special folders
  - Usually seen by the user as a single object
  - Have an identifier
  - Have a standard but flexible structure
    - Localizations live inside `*.lproj` folders
- Applications are bundles
- Frameworks are bundles
- Nibs can be bundles
  - Nibs contain UI elements
  - Nibs are created and edited with Interface Builder
  - Leopard introduced `.xib` files to hold nibs

Localizations

- Localized data are stored in an `.lproj` folder inside a bundle
  - Nibs for UI elements
  - `.strings` files for strings
- Functions look inside the various `.lproj` folder in a defined order
  - Depends on available localizations
  - Depends on the user's preferences
  - Process proceeds until a localization is found
- `.lproj` folders are named for the language they cover
  - BCP 47 is used for language IDs
    - en.lproj, fr.lproj, zh.lproj. etc.

The NSBundle Class

- Basic class to represent a bundle
- *Not* toll-free bridged with the CFBundleRef opaque type
- Getting and using bundles

```
+(NSBundle *) mainBundle
```

  - If you're an application

```
-(NSString*) pathForResource:(NSString*)name ofType:(NSString*)extension
```

  - Numerous variants are available
  - Gets the path to the best available localization for a given file

Using Locales

Locales Are…

- Objects corresponding to a particular “place”
  - Region and language are set separately
- A locale defines a default set of values and way of doing things
  - Date and time formats
  - Local currency
  - Calendar to use
- Common Locale Data Repository (CLDR)
  - Source for iPhone and Mac data

The NSLocale Class

- Core class used to represent a locale object
- Toll-free bridged with the CFLocaleRef opaque type
- Current locale assumed for many operations
  - Formatter object creation, etc.
    - Don’t have to set explicitly
- If needed, set per object
  - NSDateFormatter
    - (void)setLocale:(NSLocale *)

Creating NSLocales

`+ (NSLocale*) currentLocale`

- The locale at application launch

`+ (NSLocale*) autoupdatingCurrentLocale`

- The current locale

`NSCurrentLocaleDidChangeNotification`

`+ (NSLocale*) localeWithIdentifier:`

- Locale with an arbitrary identifier in standard form
- Identifiers for predefined locales can be listed

NSLocale Objects Give You…

- The currently set currency
- The currently set measurement system
- The currently set location
- Localized names for locale elements

```
NSLocale *locale = [ NSLocale autoupdatingCurrentLocale ];
NSString *currency = [ locale objectForKey:NSLocaleCurrencyCode ];
NSNumber *isMetric = [ locale objectForKey:NSLocaleUsesMetricSystem ];

NSString *country = [ locale objectForKey:NSLocaleCountryCode ];
// Result is something like @"US"

NSString *countryName = [ locale displayNameForKey:NSLocaleCountryCode
        value:country ];
// Result is something like @"United States"
```

Locale vs. Language

- NSLocale
  - `+ (id) currentLocale`
    - Date, time, and other settings
  - `+ (NSArray*) preferredLanguages`
    - BCP 47 language IDs
    - Ordered by user preference
- Independent of each other
  - For example, US locale with French language

Putting It Together

Putting It Together

- Getting a localized data file
- Getting a localized string
- Formatting a number

Getting a Localized File

```
- (NSString *) pathForResource:(NSString *)name
ofType:(NSString *)extension
```

```
NSBundle *bundle = [ NSBundle mainBundle ];
NSString *path = [ bundle pathForResource:@”Mountains”
            ofType:@”plist” ];
NSArray *array = [ NSArray arrayWithContentsOfFile:path ];

for ( id mountain in array ) {
  // Do something interesting
}
```

Getting a Localized String

- The `NSLocalizedString()` function

```
NSString *format = NSLocalizedString( @"sentenceFormat",
            @"A sentence with the mountain's name (first parameter), height (second
            parameter), and climbed date (third parameter)" );

NSString *name = NSLocalizedString( mount.name, nil );
```

```
NSNumberFormatter *numFormatter = [ [ NSNumberFormatter alloc ] init ];
[ numFormatter setNumberStyle:NSNumberFormatterDecimalStyle ];
NSString *height = [ numFormatter stringFromNumber:mount.height ];

NSDateFormatter *dateFormatter = [ [ NSDateFormatter alloc ] init ];
[ dateFormatter setDateStyle:NSDateFormatterMediumStyle ];
[ dateFormatter setTimeStyle:NSDateFormatterNoStyle ];
NSString *date = [ dateFormatter stringFromDate:mount.climbedDate ];

NSString *sentence = [ NSString stringWithFormat:format,
             name, height, date ];
// @"8,848 m tall, Mt. Everest was first climbed on 26 May 1953."
```

Getting a Localized String

- The `NSLocalizedString()` function

```
NSString *format = NSLocalizedString( @"sentenceFormat",
            @"A sentence with the mountain's name (first parameter), height (second
            parameter), and climbed date (third parameter)" );

NSString *name = NSLocalizedString( mount.name, nil );
```

```
NSNumberFormatter *numFormatter = [ [ NSNumberFormatter alloc ] init ];
[ numFormatter setNumberStyle:NSNumberFormatterDecimalStyle ];
NSString *height = [ numFormatter stringFromNumber:mount.height ];

NSDateFormatter *dateFormatter = [ [ NSDateFormatter alloc ] init ];
[ dateFormatter setDateStyle:NSDateFormatterMediumStyle ];
[ dateFormatter setTimeStyle:NSDateFormatterNoStyle ];
NSString *date = [ dateFormatter stringFromDate:mount.climbedDate ];

NSString *sentence = [ NSString stringWithFormat:format,
             name, height, date ];
// @"8,848 m tall, Mt. Everest was first climbed on 26 May 1953."
```

Getting a Localized String

- The `NSLocalizedString()` function

```
NSString *format = NSLocalizedString( @"sentenceFormat",
            @"A sentence with the mountain's name (first parameter), height (second
            parameter), and climbed date (third parameter)" );

NSString *name = NSLocalizedString( mount.name, nil );

NSNumberFormatter *numFormatter = [ [ NSNumberFormatter alloc ] init ];
[ numFormatter setNumberStyle:NSNumberFormatterDecimalStyle ];
NSString *height = [ numFormatter stringFromNumber:mount.height ];

NSDateFormatter *dateFormatter = [ [ NSDateFormatter alloc ] init ];
[ dateFormatter setDateStyle:NSDateFormatterMediumStyle ];
[ dateFormatter setTimeStyle:NSDateFormatterNoStyle ];
NSString *date = [ dateFormatter stringFromDate:mount.climbedDate ];

NSString *sentence = [ NSString stringWithFormat:format,
              name, height, date ];
// @"8,848 m tall, Mt. Everest was first climbed on 26 May 1953."
```

Getting a Localized String

- The `NSLocalizedString()` function

```
NSString *format = NSLocalizedString( @"sentenceFormat",
            @"A sentence with the mountain's name (first parameter), height (second
            parameter), and climbed date (third parameter)" );

NSString *name = NSLocalizedString( mount.name, nil );
```

```
NSNumberFormatter *numFormatter = [ [ NSNumberFormatter alloc ] init ];
[ numFormatter setNumberStyle:NSNumberFormatterDecimalStyle ];
NSString *height = [ numFormatter stringFromNumber:mount.height ];

NSDateFormatter *dateFormatter = [ [ NSDateFormatter alloc ] init ];
[ dateFormatter setDateStyle:NSDateFormatterMediumStyle ];
[ dateFormatter setTimeStyle:NSDateFormatterNoStyle ];
NSString *date = [ dateFormatter stringFromDate:mount.climbedDate ];

NSString *sentence = [ NSString stringWithFormat:format,
             name, height, date ];
// @"8,848 m tall, Mt. Everest was first climbed on 26 May 1953."
```

Getting a Localized String

- The `NSLocalizedString()` function

```
NSString *format = NSLocalizedString( @"sentenceFormat",
            @"A sentence with the mountain's name (first parameter), height (second
            parameter), and climbed date (third parameter)" );

NSString *name = NSLocalizedString( mount.name, nil );

NSNumberFormatter *numFormatter = [ [ NSNumberFormatter alloc ] init ];
[ numFormatter setNumberStyle:NSNumberFormatterDecimalStyle ];
NSString *height = [ numFormatter stringFromNumber:mount.height ];

NSDateFormatter *dateFormatter = [ [ NSDateFormatter alloc ] init ];
[ dateFormatter setDateStyle:NSDateFormatterMediumStyle ];
[ dateFormatter setTimeStyle:NSDateFormatterNoStyle ];
NSString *date = [ dateFormatter stringFromDate:mount.climbedDate ];

NSString *sentence = [ NSString stringWithFormat:format,
              name, height, date ];
// @"8,848 m tall, Mt. Everest was first climbed on 26 May 1953."
```

Formatting a Number

```
NSNumberFormatter *numberFormatter = [ [ NSNumberFormatter alloc ] init ];

[numberFormatter setNumberStyle:NSNumberFormatterDecimalStyle ];

NSString* formattedNumber = [numberFormatter stringFromNumber: [NSNumber
numberWithInteger: 123]];
```

Localization

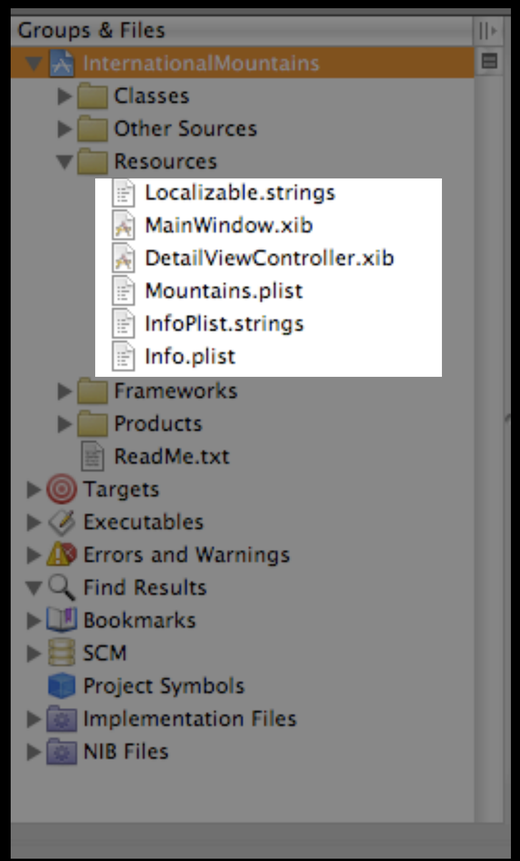
Localization Process

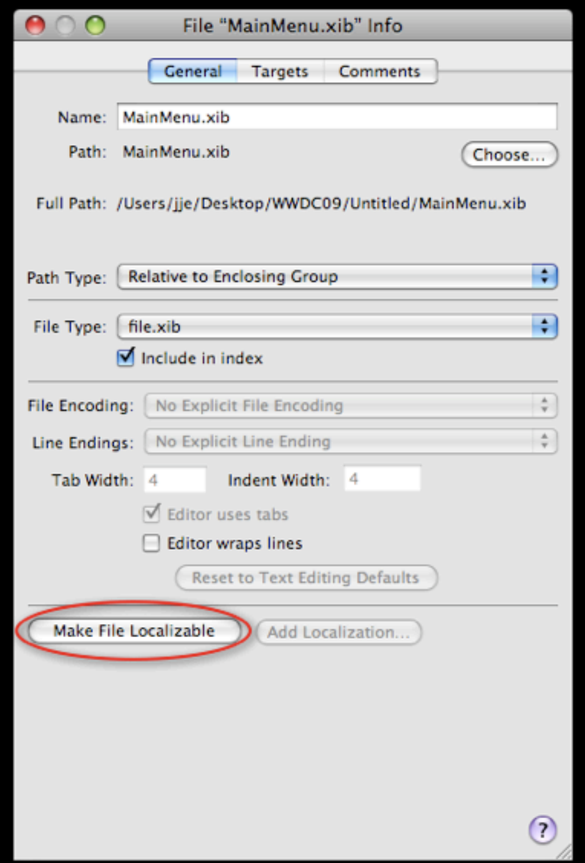
Three-step process

- Preparation of your Xcode project for localization
- Text translation
- UI layout adjustments

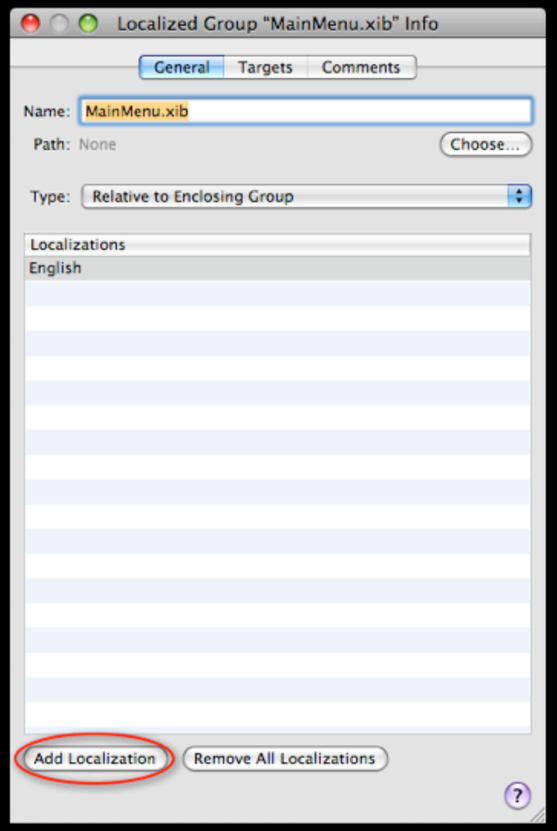
Localization Process

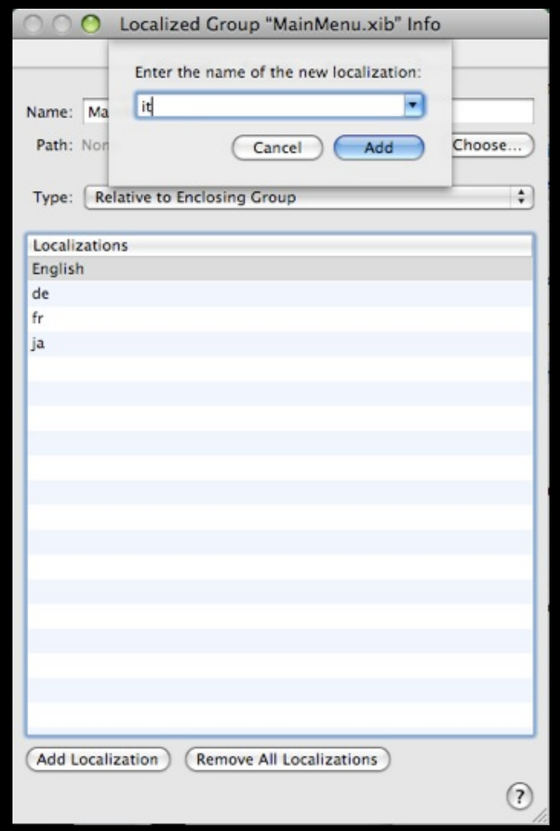
- Preparing your project for localization: "Make File Localizable"

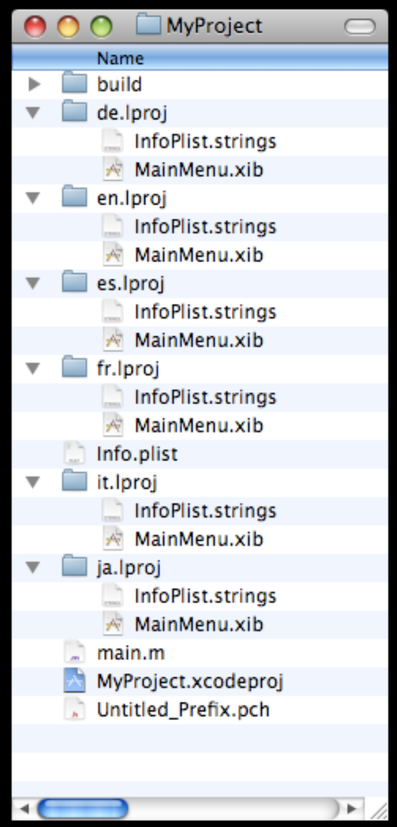




Localization Process







Localization Process

Translation process: Two approaches

- Manual—translation and UI done directly in Xcode and IB
- AppleGlot
  - Multiple file types: .strings, .plist, .nib/.xib, .xml
  - All localizable text extracted into one document
  - Supports language glossaries
  - Leverages from previous localizations
    - String matching
    - UI layout adjustments carryover

Localization Process

AppleGlot usage

```
% ./appleglot -h
% ./appleglot -d <path> create
% ./appleglot -d <path> setlangs en fr
% ./appleglot -x -d <path> populate
% ./appleglot -x -d <path> update
% ./appleglot -x -d <path> finalize
```

Localization Process

AppleGlot usage

```
% ./appleglot -h
% ./appleglot -d <path> create
% ./appleglot -d <path> setlangs en fr
% ./appleglot -x -d <path> populate
% ./appleglot -x -d <path> update
% ./appleglot -x -d <path> finalize
```

Localization Process

AppleGlot usage

```
% ./appleglot -h
% ./appleglot -d <path> create
% ./appleglot -d <path> setlangs en fr
% ./appleglot -x -d <path> populate
% ./appleglot -x -d <path> update
% ./appleglot -x -d <path> finalize
```

Localization Process

AppleGlot usage

```
% ./appleglot -h
% ./appleglot -d <path> create
% ./appleglot -d <path> setlangs en fr
% ./appleglot -x -d <path> populate
% ./appleglot -x -d <path> update
% ./appleglot -x -d <path> finalize
```

Localization Process

AppleGlot usage

```
% ./appleglot -h
% ./appleglot -d <path> create
% ./appleglot -d <path> setlangs en fr
% ./appleglot -x -d <path> populate
% ./appleglot -x -d <path> update
% ./appleglot -x -d <path> finalize
```

Localization Process

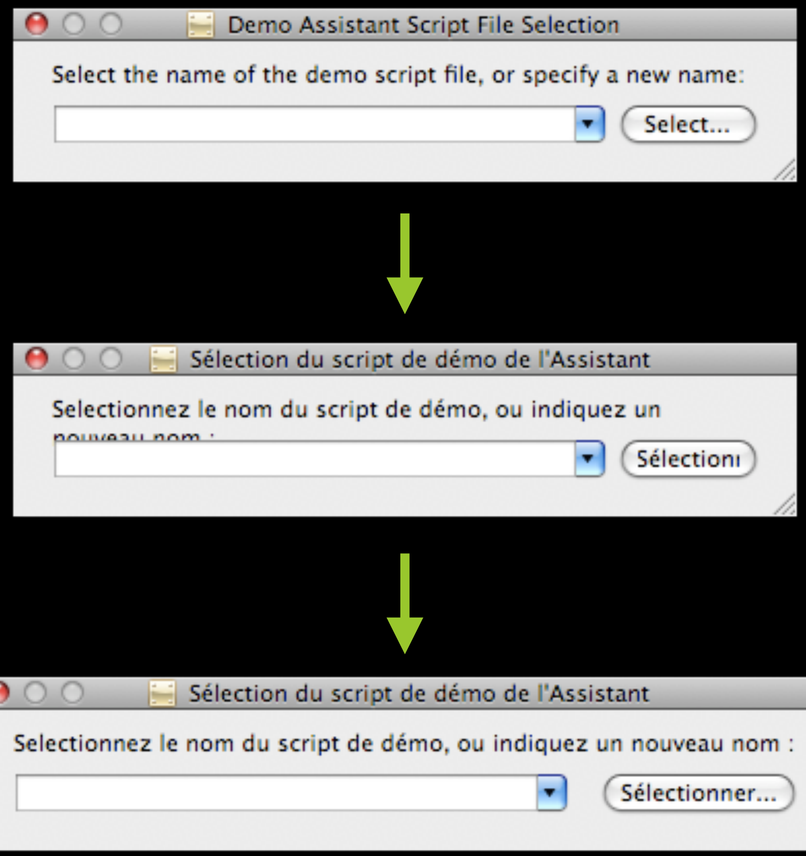
AppleGlot usage

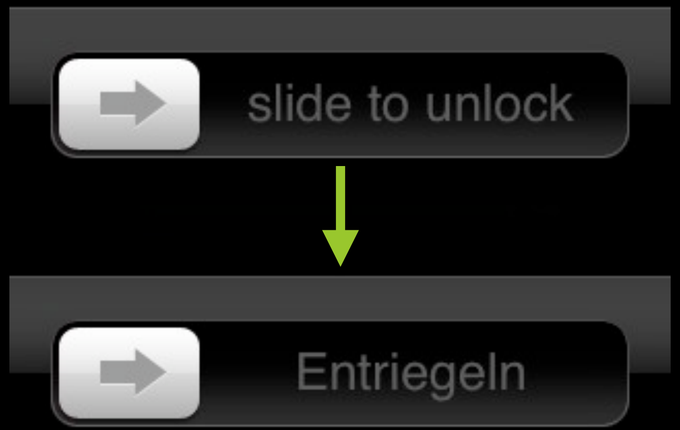
```
% ./appleglot -h
% ./appleglot -d <path> create
% ./appleglot -d <path> setlangs en fr
% ./appleglot -x -d <path> populate
% ./appleglot -x -d <path> update
% ./appleglot -x -d <path> finalize
```

Localization Process

User interface issues

- Relayout of the UI may be necessary





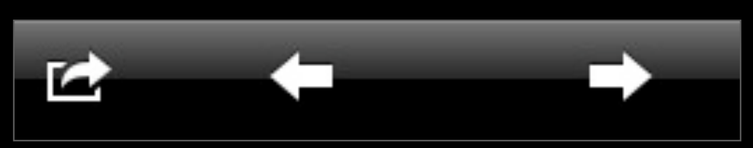
Demo

Localization

Localization—Design Tips

Do

- Allow 25% space around UI elements for text growth
- Index your variables (“%1$@”, etc.) to make reordering easy
- Use graphics and icons instead of text, wherever possible



- Verify your strings files’ integrity using xmllint or plutil

Localization—Design Tips

Don’t

- Don’t concatenate strings at runtime

```
NSMutableString *result = @"Do you really want to";
[result appendString:action];
[result appendString:@"?"];
```

-> use distinct strings or StringWithFormat

```
NSString *formatString = NSLocalizedString(@"ConfirmationString", @"comment");
NSString  *result = [NSString stringWithFormat:formatString, action];
```

- Don’t assume gender or number (“Your %@ have been saved”)

"Your preferences have been saved"

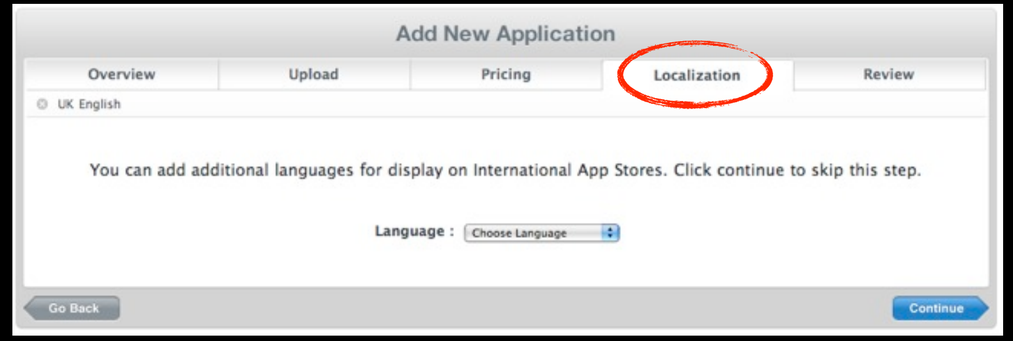
“Your investments have been saved”

“Your children have been saved”

- Don’t embed text within bitmaps

Localization and App Store Submission

- Follow Apple's SBML model, avoid MBSL (Mac OS and iPhone apps)
- If your app supports multiple languages, provide a Primary Language and add additional languages in iTunes Connect



- Provide localized metadata (product description, screen shots) in the appropriate language

Summary

Summary

- Do remember that Unicode is the key to internationalized text
- Don’t assume that text can be handled on a character-by-character basis
- Do use the APIs built into Mac OS X and iPhone OS
- Don’t try to write your own international support
- Never show hard-coded strings to the user
- Avoid assumptions about the visual relationship of UI elements or the order of words in a sentence

More Information

Matt Drance
Application Frameworks Evangelist
mdrance@apple.com

Documentation
Internationalization Programming Topics
http://developer.apple.com/documentation

XLIFF Specifications
http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html

Localization Tools
http://developer.apple.com/internationalization/localization

Related Sessions and Labs

Text Processing in Cocoa	Presidio Friday 9:00AM
Publishing on the App Store	Presidio Friday 2:00PM
Input Method Lab	OS X Lab B Friday 2:00PM
Mac and iPhone Internationalization Lab	iPhone Lab D Friday 3:30PM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09